11 | 通过程序并行计算，避免CPU资源浪费

2021-03-06 尹会生

Python自动化办公实战课 进入课程 >

讲述：尹会生
时长 19:08 大小 17.53M

你好，我是尹会生。

在我为运营工作提供技术咨询的时候，遇到过这样一个场景：这场运营活动，需要在电脑和手机端的多个不同应用程序，同时推送产品宣传图片和视频。这些大量的图片需要有不同的格式和尺寸，视频也需要根据不同的 App 截取不同的时长。

如果这类需要大量计算的多个任务成为你的日常工作，会花费你不少的时间和精力。不过别担心，我们可以通过程序并行计算，来提升任务效率。

不过你可能会说，用 Python 自动化执行，也可以提高计算效率啊，那为什么还要学习并行计算呢？

要知道，Python 默认的自动化只能利用 CPU 的一个逻辑核心，如果采用并行计算，那就能够最大化地利用 CPU 资源，从而成倍提升大量计算的任务效率。接下来我就详细分析一下并行计算的高效之处。

为什么要进行并行计算

还是我在开头提出的运营工作的场景。如果你从这类任务消耗计算机主要资源的角度去考虑，会发现这类需求有两个共同的特点。

第一，它们都需要进行大量的计算，而计算主要是通过 CPU 来实现的。CPU 的硬件指标上有两个和计算效率最相关的概念，分别是主频和多核。

主频决定 CPU 处理任务的快慢，多核决定处理的时候是否可以并行运行。这和生活中超市的收银一样，收银员的工作效率和超市开放了多少个收银台的通道，都决定了你能否以最快的速度购买到你想要买的商品。

第二，这些任务往往都需要运行相同的程序，但是程序的参数却需要根据不同的需求进行调整。

虽然咱们可以使用 Python 自动化执行这些程序，从而减少手动操作时间，但是我们还可以利用 CPU 的多核特性，让程序尽可能并行执行，发挥 CPU 的全部计算能力，提高运行效率。

那么接下来，我就来教你怎样利用 Python 的多进程库，来实现程序的并行计算，以及怎么提高并行计算的效率。

怎样实现并行计算

要想实现程序的并行计算，需要使用到标准库中的 multiprocessing 多进程库。你可能会问，进程是什么呢？

进程，是计算机用来描述程序运行状态的名词。一个进程在运行时需要消耗一定的资源，包括 CPU 的时间、内存、设备 I/O 等。如果两个进程互相独立，在同一个任务处理过程中，没有前后依赖关系，那你可以利用 multiprocessing 库同时运行多个进程，这样就能成倍地减少多个任务执行的总时间。

接下来，我就以计算 1-100 的平方为例，看看怎么使用 multiprocessing 实现并行计算。代码如下：

复制代码

```
from multiprocessing import Pool

# 计算平方
def f(x):
    return x*x

with Pool(8) as p:
    # 并行计算
    res = p.map(f, range(1, 101))
    print(f'计算平方的结果是:{res}')
```

在这段代码中，我通过 Pool 包的 map() 函数来求 1 到 100 平方计算，由于每次计算平方的过程和下一次计算没有直接关联，我就可以使用并行的方式进行计算，提高计算效率。

为了让 map() 函数能够实现并行计算，我们必须在使用它之前，通过 Pool() 包为它指定并行计算的进程数量，设置要执行的函数名称 f，以及 f() 函数所需参数。那么接下来，我就带你学习一下我是怎样使用 with 语句来设置函数的参数，并正确执行 map() 函数的。

首先来看最关键的 map() 函数，它是 Pool 包实现并行计算的函数之一。在代码中我为 map() 函数赋值了 f 和 range() 函数两个参数。

第一个参数是函数对象。

函数对象会作为 map() 函数创建进程以后，即将执行的主要任务。因此，由于这里的含义是指定 f 对象将要被创建的进程执行，而不是将 f() 函数执行的结果作为新的进程执行，所以第一个参数必须使用函数对象 f，而不能使用 f() 函数。

第二个参数要求必须是可迭代的对象。

例如我在代码中需要为 f 函数传递参数为 1-100 的整数，就可以使用 range() 函数产生 1 到 100 的整数并直接返回，因为它的返回值就是可迭代对象。

如果参数不是数字，就可以采用列表、元组、字典等支持迭代的数据类型，代替 range() 函数，作为 f() 函数的参数。举个例子，如果你需要并行调整多个视频的时长，就可以采用字典存储路径和要调整的视频时长，并把这个字典作为 map() 函数的第二个参数，map() 函数会为字典的每个键值对创建一个进程来并行处理它们。

接下来是 map() 函数中的三个主要部分，我来分析一下它们各自在并行计算中的功能。

第一，with 语句。这是我们在第七讲学习怎么使用 Python 打开文件之后，第二次用到 with 语句了。

和文件操作类似，进程打开后也需要妥善关闭，但是进程关闭属于较为底层的操作，如果你不进行操作系统层面的程序设计，是不需要对关闭进程的函数进行修改的，因为使用默认关闭进程的行为，就能满足编写并行计算的需求。

因此，multiprocessing 库对 Pool 包，支持了比较友好的进程打开和关闭方式，即 with 语句。也就是说，multiprocessing 库把对进程的操作写在 with 语句块中，而 with 语句就会自动处理进程的打开和关闭，这样在实现并行计算的代码中，你就不用学习进程的基本操作，也能减轻你学习并发程序的负担。

在了解了 with 语句可以操作进程的打开和关闭后，我们来看代码中我是怎么使用 with 语句的。

我在代码中使用了“ with Pool(8) as p ”这条语句，这里的 Pool() 类是多进程库支持的进程池功能，它的作用是指定一个多进程的程序，最多能够并行执行的进程数量。它的参数“8”，表示 map() 函数最多同时运行 8 个进程。

剩下两个部分是 range() 函数和 f() 函数。

range() 函数的作用是产生 1-100 的整数，这些整数会在每次创建新的进程时，依次作为 f() 函数的参数并赋值。而 f() 函数得到参数后，会把计算结果返回给 map() 函数。当 f() 函数处理完所有的参数后，map() 函数还会返回一个列表作为运行的结果，并进行输出。

以上就是实现并行计算的主要过程。

如何提高并行计算的效率

我们除了需要掌握并行计算的基本方法外，还可以继续提升并行计算的效率。所以在程序中还有两个地方需要优化。

一个是为并行程序自动指定并行度。在并行计算的基本方法中，我使用了手动指定并行度的方式，来指定进程最多能够运行多少个。不过手动指定的并行度并不能适合所有的电脑，因此就需要根据计算机的 CPU 核数设置合理的并行度。而且，每台计算机的 CPU 资源是固定不变的，那么设置合理的进程数量能让你的并行计算任务充分利用 CPU 资源。

另一个是统计程序运行的时间。当你对并行计算的数量做了修改后，那程序是否对计算效率起到了提升效果呢？就还需要更精确的测量，这样才能得到更准确的结果。所以我们还需要使用 Python 统计出程序执行过程一共消耗了多长的时间。

我们先来看怎么自动设置适合你的电脑的并行度。

为并行程序自动指定并行度

计算类的任务包括数字计算、数据统计、视频的编解码等，都属于计算密集型应用，它们最大的资源开销就是 CPU 时间的消耗，设置的并行度过大或过小都不能达到最好的运行效率。

> 如果并行度设置过小，比如运行的进程数量小于逻辑 CPU 的数量，就会造成部分逻辑 CPU 因为无法被充分利用而处于闲置状态。
>
> 如果并行度设置过大，由于现代的操作系统为了保证每个进程都能公平得到 CPU 资源，所以会造成 CPU 把时间大量消耗在进程切换上。那么并行度设置过大，会导致 CPU 还未完成一个进程的处理时，就得切换至下一个进程进行处理，而多进程之间来回切换也会消耗 CPU 时间，造成 CPU 资源的浪费。

那并行度该怎么设置才合理呢？通常情况下，我们会把并行度设置为逻辑 CPU 数量的两倍。不过假如计算任务达到小时级别（这类任务需要长时间占用 CPU 资源），为了减少切换任务时的开销，我建议计算的并行度和逻辑 CPU 数量保持相等。

这就又有一个问题了，该怎么获得计算机的逻辑 CPU 个数呢？Windows 可以通过任务管理器获得，MacOS 可以通过活动监视器获得。如果你希望取得逻辑 CPU 的个数之后，可

以根据它的数量自动设置创建进程的数量，那么可以通过安装第三方包 psutils，利用其中的 cpu_count() 函数取得逻辑 CPU 个数。

我把并行度自动设置为当前逻辑 CPU 两倍的代码写在下面，供你参考。

复制代码

```
from multiprocessing import Pool,Queue
import os
import psutil

# 逻辑cpu个数
count = psutil.cpu_count()

# 定义一个队列用于存储进程id
queue = Queue()

# 用于计算平方和将运行函数的进程id写入队列
def f(x):
    queue.put(os.getpid())
    return x*x

with Pool(count*2) as p:
    # 并行计算
    res = p.map(f, range(1, 101))
    print(f'计算平方的结果是:{res}')

# 并行计算用到的进程id
pids = set()
while not queue.empty():
    pids.add(queue.get())

print(f'用到的进程id是: {pids}')
```

在代码中，我使用了 psutil.cpu_count() 函数来获取逻辑 CPU 的个数，它把“count*2”作为参数传递给 Pool() 类，并以逻辑 CPU 两倍作为最大创建进程数量，从而计算 1-100 的平方。

这里有两点需要你注意。第一，**psutils 是 process and system utilities** 的缩写，所以它除了获取逻辑 CPU 数量外，还可以获取内存、硬盘、网络连接等和操作系统相关的信息。如果你在工作中需要取得操作系统的运行状态，就可以采用 psutils 包。

第二，psutils 是第三方库，因此，在 Windows 上你需要通过 cmd 命令行执行 pip3 install psutil 安装后，才能释放 psutils 包，否则会出现模块无法找到的错误。

由于 map() 函数的第二个参数可能会被传入不可迭代对象，这时有可能会导致只运行了一个进程，因此我就在多进程执行过程中，增加了记录进程 ID 的功能。而在这一功能中，我使用的是 **os 库、队列库和集合数据类型，**按照下面三个步骤来实现对所有创建的进程 ID 的统计。

首先，使用os 库的 getpid() 函数获取进程 ID。

由于 map() 函数会根据 Pool() 类的参数，事先创建好指定数量的进程，而每次运行 f() 函数都在创建好的进程中执行，所以我就采用 os 库的 getpid() 函数取得运行 f() 函数进程的唯一标识，这就是使用 os 库的用途。

接下来，使用队列库存储每次运行进程的 ID。

为了把每次运行的进程 ID 存到一个对象中，我使用了 multiprocessing 库的队列包。因为在多进程的程序中，不能采用标准数据类型来传递数据，所以 multiprocessing 库还提供了方便进程间通信的对象——Queue 队列。

map() 函数每执行一次 f() 函数，我就把进程 ID 作为队列的 put() 函数的参数，并把进程 ID 放入队 Queue 中，直到所有的 f() 函数执行完成，队列里就会记录每次执行的进程 ID 信息。

最后，使用集合数据类型存储本次 f() 函数运行的所有进程 ID。

为了实现这一功能，我需要通过 while 循环结构，根据队列不为空的条件，把队列中的进程 ID 使用 get() 函数取出来，放入 pids 变量中。

pids 变量是集合数据类型，集合是一个无序的不重复元素序列，需要使用 set() 创建。你可以把集合当作一个只有键没有值的字典来记忆，它的特点是集合里的元素不能重复。

由于 f() 函数会多次在一个进程中执行，因此在队列中会记录重复的进程 ID，我把进程 ID 从队列中取出后，放入集合数据类型中，自己就不用编写程序，自动把重复的进程 ID 去掉

了。而且通过对集合 pids 中的进程 ID 进行输出，可以看到进程 ID 的数量刚好和 Pool() 类指定的并行进程数量相等。

这种用法是我经常在进行多进程程序调试的一种简单用法，我还会把它们的结果写入文件保存，以便程序出现异常执行结果时，可以根据调试的信息进行问题的定位。

统计程序运行的时间

我们除了需要掌握判断程序的并行度外，还可以统计并行计算比顺序计算节省了多少时间。那么再遇到相同场景的时候，你可以选择并行方式来运行程序，提高工作效率。接下来我来教你怎样统计 Python 程序运行的时间。

在 Python 中我们可以利用 time 库的 time() 函数，来记录当前时间的功能。

> 首先，需要在统计时间代码的前后各增加一次 time.time() 函数，并把它们统计时间的结果存放在 time1、time2 两个不同的变量中。
>
> 然后再把两个变量相减，这样就能取得程序的运行时间了。

我把核心实现代码写在下面供你参考。

复制代码

```
# 并行计算时间统计
with Pool(4) as p:
    # 并行计算
    time1 = time.time()
    res = p.map(f, range(1, 10001))
    time2 = time.time()
    # print(f'计算平方的结果是:{res}')

print(str(time2-time1))


# 串行计算时间统计
list1 = []

time1 = time.time()
for i in range(1, 10001):
    list1.append(f(i))
time2 = time.time()

print(str(time2-time1))
```

在这段代码中，通过 time1 和 time2 的时间差就可以得到程序运行的时间了，那么根据运行时间，我们可以把并行程序和串行程序执行时间的性能进行对比。

这里你需要注意，由于计算平方的 CPU 开销较小，比较难体现并行计算的优势，你就可以采用并行访问网页，或其他 CPU 开销较高的程序，这样会让两个程序的时间差别更加明显。

总结

在最后，我来为你总结一下实现并行计算的**基本方法和三个注意事项。**

通过 multiprocessing 的 Pool 包可以实现基于进程的并行计算功能，Pool 包的 map() 函数会根据 Pool 包指定的进程数量实现并行运行。这里还有三点需要你注意：

1. 作为 map() 函数的第一个参数你需要传递函数对象 f，不能传递函数的调用 f() 形式，这是初学者实现并行任务最容易出现的错误。
2. 为了让并行度更适合你的电脑，应该根据逻辑 CPU 的个数设置并行度，并根据运行时间来对并行数量进一步优化。
3. 实现并行计算任务的程序除了使用多进程模型外还可以使用多线程模型。多进程的并行计算更适用于计算密集型应用，即程序运行过程中主要为计算类 CPU 开销大的程序，多线程模型适合 I/O 密集型的应用，例如: 通过互联网进行批量网页访问和下载。如果你想将多进程的并发模型改为多线程的并发模型只需在导入库的时候
将“multiprocessing”改为“multiprocessing.dummy”就能实现多线程并行访问网页。我将多进程和多线程两种方式导入库的代码贴在下方供你参考。

复制代码

```
# 多进程模型
from multiprocessing import Pool

# 多线程模型
from multiprocessing.dummy import Pool

# multiprocessing.dummy的Pool用法和multiprocessing库相同
```

我把这节课的相关代码放在了GitHub上，你可以自行查找、学习。

思考题

我为你留一道思考题，有一个软件包 requests，可以通过 requests.get('http://www.baidu.com').text 方式访问一个网站，并能够得到网页的源代码。假设我为你提供了几十个需要访问的网站，你是如何实现这些网站的并行访问的，你又能否通过 Python 对比出逐个访问网页的时间是并行访问的几倍吗？

提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。 页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 10 | 按指定顺序给词语排序，提高查找效率

下一篇 12 | 文本处理函数：三招解决数据对齐问题

精选留言 (2)

写留言

十一哈哈
2021-03-08

windows下使用multiprocessing，要将进程池相关代码应该放在if __name__ == '__main__'下面，要不然运行会报错....

展开

作者回复: 是的，感谢，忘记在文字下方提醒大家。

2

Soul of the Drago...
2021-03-06

老师，请问如果在代码运行过程中出现“UnicodeEncodeError: 'ascii' codec can't encode characters in position 18-19: ordinal not in range(128)” 这样的报错，应该如何解决呢?

作者回复: 你的Python应该是python2.x版本，需要将它换成Python3版本，要从官方网站重新下载。